S6800 TINY BASIC

AMI

American Microsystems, Inc. • 3800 Homestead Road • Santa Clara CA 95051

AMI S6800 TINY BASIC

The programming language, BASIC ("Beginner's All-purpose Symbolic Instruction Code"), was originally developed at Dartmouth as a language to teach programming, and has since become one of the most popular languages for all applications because of its simplicity, generality, and ease of use. Tiny Basic was likewise defined as a learning tool at the People's Computer Company, and has similarly gained considerable popularity because of its small size and minimal computer requirements.

Tiny Basic is a subset of the original Dartmouth Basic, omitting a few of the more esoteric statement types, strings, and arrays, and limiting the user to only 26 variables (the single letters of the alphabet) and 16-bit integers. To this is added a computed line number capability for the GOTO and GOSUB statements, so that the language is both complete and powerful.

The AMI Prototyping Board version of Tiny Basic is designed to execute entirely on an unmodified PROTO system. The Tiny Basic interpreter executes from the EPROM space, E000-E7FF, using the four S6834 sockets on the Proto board. The interpreter temporary storage, and the user data variables are stored in memory page 00, and all the rest of the memory space is available to the user for Basic program storage and the GOSUB return address stack. Input and output using the Basic INPUT and PRINT statements is done using the S6850 ACIA connected to the user teleprinter or other terminal. Direct byte input and output is possible using the USR function, with reference to the S6820 PIAs on the system, although Tiny Basic makes no direct reference to them.

The AMI S6800 Tiny Basic is designed to use all of the memory space (RAM) available to it for user programs. When the program is initially started from Proto with a G E000 command, the Tiny

Basic interpreter scans the memory space, beginning with the second half of Page 00. All of the RAM (writable) memory contiguous with page 00 is allocated as part of the user program space. If the system has no added memory, location 0200 will be found to be vacant in this scan, so Tiny Basic allocates the two pages of RAM in the upper address space for program memory (i.e. FE00-FFF7) also, a total of a little more than 800 bytes of memory. On the other hand, if the user has added large amounts of memory in the lower address space, so that it is contiguous with page 00, Tiny Basic will use only that part of the memory for user program and stack space, leaving the upper RAM untouched. Note that Tiny Basic uses nearly all of memory page 00 (addresses 0000-00FF), and this memory must be present for Tiny Basic to function.

To run Tiny Basic requires only that the EPROMs be installed, and the Proto command,

```
>G E000
```

be typed in. Since Tiny Basic does not initialize the ACIA, the restart vector switches should not be directly set to E000 without going through Proto at least once after power on. The High/Low RAM switch should be set to "Low" unless additional memory has been placed in the system beginning at location 0000. When properly entered, Tiny Basic will type out the command line prompt, a colon, on the left margin of a new line. Now command lines or numbered program statements may be typed in.

All input to Tiny is buffered in a 72 character line, terminated by a Carriage Return ("CR"). Excess characters are ignored, as signalled by ringing the console/terminal bell. When the CR is typed in, Tiny will echo it with a Linefeed, then proceed to process the information in the line. If a typing error occurs during the input of either a program line or data for an INPUT statement, the erroneous characters may be deleted by "backspacing" over them and retyping. If the entire line is in error, it may be

cancelled (and thus ignored) by typing the "Cancel" key. The Backspace code is a "left arrow" or ASCII Underline (shift 0 on your Teletype). The Cancel code is the ASCII Cancel code (Control X).

When Tiny ends a line (either input or output), it types a CR, two pad characters, a Linefeed, and one more pad character. The pad character used is a NULL code (hex 00).

Tiny Basic 6800 has a provision for suppressing output (in particular line prompts) when using paper tape for loading a program or inputting data. This is activated by the occurrence of a Linefeed in the input stream (note that the user normally has no cause to type a Linefeed since it is echoed in response to each CR), and disables all output (including program output) until the tape mode is deactivated. The tape mode is turned off by the occurrence of an X-OFF character (ASCII DC3, or Control S) in the input, by the termination of an executing program due to an error, or after the execution of any statement or command which leaves Tiny in the command mode.

Occasionally it is desirable to interrupt the operation of a Tiny Basic program, or to abort a long program listing. This may be done by typing the Escape key on the terminal. Actually any key (except NUL) may be typed, and program execution will be suspended at the completion of the current statement line. Note that the printout from a single PRINT statement cannot be interrupted this way, but when that statement completes, and execution is about to advance to the next statement, the escape keyin typed during the printout will be honored.

Under certain circumstances a programming error in the Basic program may have catastrophic consequences, requiring the MPU to be reset. Although this is only possible in connection with improper use of the USR function, if the program is long, it may be desirable to recover without destroying the program stored in the user program space. For this purpose only, Tiny Basic may be entered with the Proto command,

```
>G E003
```

which bypasses the initialization of the memory, so that the program is preserved. The user should note, however, that if the catastrophic failure also destroyed certain parameters in page 00, recovery by this means may not be possible.

STATEMENTS

Tiny Basic 6800 is a subset of Dartmouth Basic, with a few extensions to adapt it to the microcomputer environment. Appendix B contains a BNF definition of the language; the discussion here is intended to enable you to use it. When Tiny issues a line prompt (a colon on the left margin) you may type in a statement with or without a line number. If the line number is included, the entire line is inserted into the user program space in line number sequence, without further analysis. Any previously existing line with the same line number is deleted or replaced by the new line. If the new line consists of a line number only, it is considered a deletion, and nothing is inserted. Blanks are not significant to Tiny, so blanks imbedded in the line number are ignored; however, after the first non-blank, non-numeric character in the line, all blanks are preserved in memory.

The following are valid lines with line numbers:

```
123 PRINT "HELLO"
456    G O T O 1 2 3
7 8 9  PRINT "THIS IS LINE #789"
123
32767 PRINT "THIS IS THE LARGEST LINE #"
1PRINT"THIS IS THE SMALLEST LINE#"
10000 TINY BASIC 6800 DOES NOT CHECK
10001 FOR EXECUTABLE STATEMENTS ON INSERTION.
```

0 is not a valid line number.

If the input line does not begin with a line number, it is executed directly, and must consist of one of the following statement types:

LET	GOTO	REM
IF . . . THEN	GOSUB	CLEAR
INPUT	RETURN	LIST
PRINT	END	RUN

These statement types are discussed in more detail in the pages to follow.

Note that all twelve statement types may be used in either the Direct Execution mode (without a line number) or in a program sequence (with a line number). Two of the statements (INPUT and RUN) behave slightly differently in these two operating modes, but otherwise each statement works the same in Direct Execution as within a program. Obviously there is not much point in including such statements as RUN or CLEAR in a program, but they are valid. Similarly, a GOSUB statement executed directly, though valid, is likely to result in an error stop when the corresponding RETURN statement is executed.

EXPRESSIONS

Many of these statement types involve the use of EXPRESSIONS. An expression is the combination of one or more NUMBERS or VARIABLES, joined by OPERATORS, and possibly grouped by Parentheses. There are four Operators:

+	addition
-	substraction
*	multiplication
/	division

These are hierarchical, so that in an expression without parentheses, multiplication and division are performed before addition and substraction. Similarly, sub-expressions within parentheses are evaluated first. Otherwise evaluation proceeds from left to right. Unary operators (+ and -) are allowed in front of an expression to denote its sign.

A number is any sequence of decimal digits (0, 1, 2,...9), denoting the decimal number so represented. Blanks have no significance and may be imbedded within the number for readability

if desired, but commas are not allowed. All numbers are evaluated as 16-bit signed numbers, so numbers with five or more digits are truncated modulo 65536, with values greater than 32767 being considered negative.

The following are some valid numbers (note that the last two are equivalent to the first two in TINY):

```
0
100
10 000
1 2 3 4
32767
65536
65 536
```

A Variable is any Capital letter (A, B, ...Z). This variable is assigned a fixed location in memory (two bytes, the address of which is twice the ASCII representation of the variable name). It may assume any value in the range, -32768 to +32767, as assigned to it be a LET or INPUT statement.

The following are some examples of valid expressions:

```
A
123
1+2-3
B-14*C
(A+B)/(C+D)
-128/(-32768+(I*I)
(((((Q)))))
```

All expressions are evaluated as integers modulo 65536. Thus an expression such as

```
N / P * P
```

may not evaluate to the same value as (N),and in fact this may be put to use to determine if a variable is an exact multiple of some number. TINY BASIC 6800 also makes no attempt to discover arithmetic overflow conditions, except in the case of an attempt to

divide be zero (which results in an error stop). Thus all of the following expressions evaluate to the same value:

```
-4096
15*4096
32768/8
30720+30720
```

TINY BASIC 6800 allows two intrinsic functions. These are:

Random Number Generator
User Sub-routine

Either of these functions may be used anywhere an expression is appropriate.

FUNCTIONS

RANDOM NUMBER Generation

RND (range)

This function has as its value, a positive pseudo-random number between zero and range -1, inclusive. If the range argument is zero an error stop results.

USER SUB-ROUTINE

USR (address)
USR (address, Xreg)
USR (address, Xreg, A&Breg)

This function is actually a machine-language subroutine call to the address in the first argument. The second argument is included; the 6800 X register contains that value on entry to the subroutine. If the third argument is included, the 6800 A and B accumulators contain that value on entry to the subroutine, with the most significant part in B. On exit, the value in the A and B accumulators becomes the value of the function. All three arguments are evaluated as normal expressions (numbers are decimal). For your convenience, location -8172 (decimal, hexadecimal address equivalent E014 is the entry point of a subroutine to read one byte from the memory address in the X register, and location

-8168 is the entry point of a subroutine to store one byte into memory. Appendix E gives examples of the USR function used for memory access.

STATEMENT TYPES

PRINT print-list
PR print-list

This statement prints on the console/terminal the values of the expressions and/or the contents of the strings in the print-list. The print-list has the general form,

item, item. . . or item; item. . .

The items may be expressions or alphanumeric strings enclosed in quotation marks (e.g. "string"). Expressions are evaluated and printed as signed numbers; strings are printed as they occur in the PRINT statement. When the items are separated by commas the printed values are justified in columns of 8 characters wide; when semicolons are used there is no separation between printed items. Thus,

```
PRINT 1,2,3
```

prints as

```
1       2       3
```

and

```
PRINT 1;2;3
```

prints as

```
123
```

Commas and semicolons, strings and expressions may be mixed in one PRINT statement at will.

If a PRINT statement ends with a comma or semicolon TINY BASIC 6800 will not terminate the output line so that several PRINT statements may print on the same output line, or an output message may be printed on the same line, or an output message may be printed on the same line as an input request (see INPUT). When the PRINT statement does not end with a comma or semicolon the output is terminated with a carriage return and linefeed (with their associated pad characters). To aid in preparing data tapes for input to other programs, a colon at the end of a print-list will output an "X-OFF" control character just before the Carriage Return.

Although the PRINT statement generates the output immediately while scanning the statement line, output lines are limited to 127 characters, with excess suppressed.

While the Break Key will not interrupt a PRINT statement in progress, the Break condition will take effect at the end of the current PRINT statement.

The following are some examples of valid PRINT statements:

```
PRINT "A="; A, "B+C="; B+C
PR                              (one blank line)
PRI                             (prints the value of I)
PRINT 1,",",Q*P;",",R/42:
```

INPUT input-list

This statement checks to see if the current input line is exhausted. If it is, a question mark is prompted with an X-ON control character, and a new line is read in. Then or otherwise, the input line is scanned for an expression which is evaluated. The value thus derived is stored in the first variable in the input-list, the process is repeated. In an executing program, several values may be input on a single request be separating them with commas. If these values are not used up in the current INPUT statement they are saved for subsequent INPUT statements. The question mark is prompted only when a new line of input values is required. Note that each line of input values must be terminated by a carriage return. Since expressions may be used as input values, any letter in the input line will be interpreted as the value of that variable. Thus if a program sets the value of A to 1, B to 2, and C to 3, and the following statement occurs during execution:

```
INPUT X,Y,Z
```

and the user types in

```
A,C,B
```

the values entered into X,Y, and Z will be 1, 3, and 2, respectively, just as if the numbers had been typed in. Note also that

blanks on the input line are ignored by TINY, and the commas are required only for separation in cases of ambiguity. In the example above,

```
ACB
```

could have been typed in with the same results. However an input line typed in as

```
+1  -3  +6  0
```

will be interpreted by TINY as a single value (=58) without commas for separators.

Due to the way TINY BASIC 6800 buffers its input lines, the INPUT statement cannot be directly executed for more than one variable at a time, and if the following statement is typed in without a line number,

```
INPUT A,B,C
```

the value of B will be copied to A, and only one value (for C) will be requested from the console/terminal. Similarly, the statement,

```
INPUT X,1,Y,2,Z,3
```

will execute directly (loading X,Y, and Z with the values 1,2, and 3), requesting no input, but with a line number in a program this statement will produce an error stop after requesting one value.

If the number of expressions in the input line does not match the number of variables in the INPUT statement, the excess input is saved for the next INPUT statement, or another prompt is issued for more data. The user should note that misallignment in these circumstances may result in incorrect program execution (the wrong data to the wrong variables). If this is suspected, data entry may be typed in one value at a time to observe its synchronization with PRINT statements in the program.

There is no defined escape from an input request, but if an invalid expression is typed (such as a period or a pair of commas) an invalid expression error stop will occur.

Because TINY BASIC does not allow arrays about the only way to process large volumes of data is through paper tape files. Each input request prompt consists of a question mark followed by an X-ON (ASCII DC1) control character to turn on an automatic paper tape reader on the Teletype (if it is ready). A paper tape may be prepared in advance with data separated by commas, and an X-OFF (ASCII DC3 or Control -5) control character preceding the CR (a Teletype will generally read at least one more character after the X-OFF). In this way the tape will feed one line at a time, as requested by the succession of INPUT statements. This tape may also be prepared from a previous program output (see the PRINT statement).

LET var = expression
var = expression

This statement assigns the value of the expression to the variable (var). The following are valid LET statements:

```
LET A = B+C
I = 0
LET Q = (RND (33) +5)
```

GOTO expression

The GOTO statement permits changes in the sequence of program execution. Normally programs are executed in the numerical sequence of the program line numbers, but the next statement to be executed after a GOTO has the line number derived by the evaluation of the expression in the GOTO statement. Note that this permits you to compute the line number of the next statement on the basis of program parameters during program execution. An error stop occurs if the evaluation of the expression results in a number for which there is no line. If a GOTO statement is executed directly, it has the same effect as if it were the first line of a program, and the RUN statement were typed in, that is, program

execution begins from that line number, even though it may not be the first in the program. Thus a program may be continued where it left off after correcting the cause of an error stop. The following are valid GOTO statements:

```
GOTO 100
GO TO 200+I*10
G O T O X
```

GOSUB expression

The GOSUB statement is like the GOTO statement,except that TINY remembers the line number of the GOSUB statement, so that the next occurrance of a RETURN statement will result in execution proceeding from the statement following the GOSUB. Subroutines called by GOSUB statements may be nested to any depth, limited only by the amount of user program memory remaining. Note that a GOSUB directly executed will result in an error stop at the corresponding RETURN. The following are some examples of valid GOSUB statements:

```
GOSUB 100
GO SUB 200+I*10
```

RETURN

The RETURN statement transfers execution control to the line following the most recent unRETURNed GOSUB. If there is no matching GOSUB an error stop occurs.

IF expression rel expression THEN statement
IF expression rel expression statement

The IF statement compares two expressions according to one of six relational operators. If the relationship is True, the statement is executed; if False, the associated statement is skipped. The six relational operators are:

```
=     equality
<     less than
```

>	greater than
< =	less or equal (not greater)
> =	greater or equal (not less)
<>, ><	not equal (greater or less)

The statement may be any valid TINY BASIC 6800 statement (including another IF statement). The following are valid IF statements:

```
IF I>25 THEN PRINT "ERROR"
IF N/P*P=N GOTO 100
IF I=2 Then this is nonsense
IF RND (100) >50 THEN IF I <>  J INPUT Q,R
```

END

The END statement must be the last executable statement in a program. Failure to include an END statement will result in an error stop after the last line of the program is executed. The END statement may be used to terminate a program at any time, and there may be as many END statements in a program as needed. The END statement also clears out any saved GOSUB line numbers remaining, and may be used for that purpose in the direct execution mode.

REM Comments

The REM statement permits comments (remarks) to be interspersed in the program. Its execution has no effect on program execution, except for the time taken.

CLEAR

The CLEAR statement formats the user program space, deleting any previous programs. If included in a program (i.e. with a line number) the program becomes suicidal when the statement is executed, although no error results.

RUN
RUN, Expression-list

The RUN statement is used to begin program execution at the

first (lowest) line number. If the RUN statement is directly executed, it may be followed by a comma, followed by values to be input when the program executes an INPUT statement.

If the RUN statement is included in a program with a line number, its execution works like a GO TO first statement of the program.

LIST
LIST Expression
LIST Expression, Expression

The LIST statement causes part or all of the user program to be listed. If no parameters are given, the whole program is listed. A single expression parameter is evaluated to a line number which, if it exists, is listed. If both expression parameters are given, all of the lines with line numbers between the two values (inclusive) are listed. If the last expression in the LIST statement evaluates to a number for which there is no line, the next line above that number which does exist (if any) is listed as the last line. Zero is not a valid line number, and an error stop will occur if one of the expressions evaluates to zero. A LIST statement may be included as part of the program, which may be used for printing large text strings such as instructions to the operator. A listing may be terminated by the Break key. The following are valid LIST statements:

```
LIST
LIST 75 + 25     (lists line 100)
LIST 100,200
LIST 400,300     (lists nothing)
```

A P P E N D I X A

ERROR MESSAGE SUMMARY

0	Break during execution
8	Memory overflow; line not inserted
9	Line number 0 not allowed
13	RUN with no program in memory
18	LET is missing a variable name
20	LET is missing a =
23	Improper syntax in LET
25	LET is not followed by END
34	Improper syntax in GOTO
37	No line to GO TO
39	Misspelled GOTO
40,41	Misspelled GOSUB
46	GOSUB subroutine does not exist
59	PRINT not followed by END
62	Missing close quote in PRINT string
73	Colon in PRINT is not at end of statement
75	PRINT not followed by END
95	If not followed by END
104	INPUT suntax bad - expects variable name
123	INPUT syntax bad - expects comma
124	INPUT not followed by END
132	RETURN suntax bad
133	RETURN has no matching GOSUB
134	GOSUB not followed by END
139	END syntax bad
154	Can't LIST line number 0
164	LIST syntax error - expects comma
183	REM not followed by END
184	Missing statement type keyword
186	Misspelled statement type keyword
188	Memory overflow: too many GOSUB's . . .

211 . . . or expression to complex
224 Divide by 0
226 Memory overflow
232 Expression too complex . . .
233 . . . using RND . . .
234 . . . in direct evaluation;
252 . . . simplify the expression
259 RND (0) not allowed
266 Expression too complex
267 . . . for RND
275 USR expects "(" before arguments
284 USR expects ")" after arguments
287 Expression too complex . . .
288 . . . for USR
290 Expression too complex
304 Memory overflow (in function evaluation)
306 Syntax error - expects "(" for function arguments
330 IF syntax error - expects relation operator

A P P E N D I X B

FORMAL DEFINITION OF TINY BASIC 6800

```
line : := number statement CR
          statement CR
statement ::= PRINT printlist
              PR printlist
              INPUT varlist
              LET var = expression
              var = expression
              GOTO = expression
              GOSUB expression
              RETURN
              IF expression relop expression THEN statement
              IF expression relop expression statement
              REM commentstring
              CLEAR
              RUN
              RUN exprlist
              LIST
              LIST exprlist
printlist ::=
              printitem
              printitem
              printitem separator printlist
printitem ::= expression
              "characterstring"
varlist ::= var
            var, varlist
esprlist ::= expression
             expression, exprlist
expression ::= unsignedexpr
               + unsignedexpr
               - unsignedexpr
```

```
term ::= factor
         factor * term
         factor / term
factor ::= var
           number
           ( expression )
           function
function ::= RND ( expression )
             USR ( exprlist )
number ::= digit
          digit number
separator ::= , ! ;
var ::= A ! B ! ... ! Y ! Z
digit ::= 0 ! 1 ! 2 ! ... ! 9
relop ::= < !  > ! = ! <= ! >=! <> ! ><
```

A P P E N D I X C

MAKING EROMS FROM PAPER TAPE

If you received Tiny Basic in the form of a Paper Tape, the contents of the tape must be copied onto four S6834 Erasable ROMS before the program can be used. These are prepared in the following way:

Place the tape in the reader of your Teletype with the RAM switch set for high memory, and type

```
>L FC00-FDFF+1C00
```

When the Teletype ball begins to bounce, stop the reader, type an Escape, then insert an EROM in the Burner socker and type

```
>B
```

When Proto comes back with a caret prompt (about 45 seconds), remove the EROM and label it "E0". Back the paper tape in the reader up about eight inches (over the previous line break), and type

```
>L FC00-FDFF+1A00
```

Within ten seconds after starting the reader the Teletype ball should bounce twice, then loading continues for the second EROM. If the ball bounces about 30 times, you backed the tape too far; this does not hurt anything. If it does not bounce at all (except at the line breaks), you did not back up far enough; stop the reader, type an ESCape, and try again, pulling the tape back through farther than before. When the teletype ball begins to bounce again, stop the teader, and burn the EROM to be labeled "E1". Repeat also for +1800 ("E2") and +1600 ("E3"). At the end of the tape (i.e. while loading for the last EROM), the reader will stop itself, and Proto will type

```
EOF
>
```

This is normal, and you may proceed directly to burn the last EROM. The EROM's are then plugged into the sockets in the Proto board

(positions 6-9 respectively), the RAM switch set to "low", and the command typed,

```
>G E000
```

A P P E N D I X D

LOW MEMORY MAP

LOCATION	SIGNIFICANCE
0000-000F	Not used by Tiny
0010-001D	Proto version temporaries
001E-001F	1K split RAM flag (0200=split)
0020-0021	Lowest address of user program space
0022-0023	Highest address of program space
0024-0025	Program end + stack reserve
0026-0027	Top of GOSUB stack
0028-002F	Interpreter parameters
0030-007F	Input line buffer & Computation stack
0080-0081	Random Number Generator workspace
0082-0083	Variable "A"
0084-0085	Variable "B"
...	...
00B4-00B5	Variable "Z"
00B6-00C7	Interpreter temporaries
00B8	Start of User program (PROTO)
E000	Cold Start entry point
E003	Warm Start entry point
E006 08	JMP (or JSR) to character input
E009 0B	JMP to character output
E00C 0E	JMP to Break test
E00F	Backspace code
E010	Line Cancel code
E011	Pad character
E012	Tape Mode Enable flag (hex 80 = enabled)
E013	Spare stack size
E014	Subroutine to read one Byte from RAM to B&A (address in X)
E018	Subroutine to store A&B into RAM at address in X

A P P E N D I X E
AN EXAMPLE PROGRAM

```
10   REM DISPLAY 64 RANDOM NUMBERS < 100 ON 8 LINES
20   LET I = 0
30   PRINT RND (100)
40   LET I=I+I
50   IF I/8*8=I THEN PRINT
60   IF I<64 THEN GOTO 30
70   END


100  REM PRINT HEX MEMORY DUMP
109  REM INITIALIZE
110  A=-10
120  B=-11
130  C=-12
140  D=-13
150  E=-14
160  F=-15
170  X = -1
200  REM GET (HEX) ADDRESSES
210  PRINT "DUMP:  L,U";
215  REM INPUT STARTING ADDRESS IN HEX
220  GOSUB 500
230  L=N
235  REM INPUT ENDING ADDRESS IN HEX
240  GOSUB 500
250  U=11
275  REM TYPE OUT ADDRESS
280  GOSUB 450
290  REM GET MEMORY BYTE
300  LET N = USR (-8172,L)
305  REM CONVERT IT TO HEX
```

```
310  LET M = N/16
320  LET N = N-M*16
330  PRINT " ";
335  REM PRINT IT
340  GOSUB 400+M+M
350  GOSUB 400+N+N
355  REM END?
360  IF L=U GO TO 390
365  L=K+1
370  IF L/16*16 = L GOTO 280
375  REM DO 16 BYTES PER LINE
380  GO TO 300
390  PRINT
399  PRINT ONE HEX DIGIT
400  PRINT 0;
401  RETURN
402  PRINT 1;
403  RETURN
404  PRINT 2;
405  RETURN
406  PRINT 3;
407  RETURN
408  PRINT 4;
409  RETURN
410  PRINT 5;
411  RETURN
412  PRINT 6;
413  RETURN
414  PRINT 7;
415  RETURN
416  PRINT 8;
417  RETURN
418  PRINT 9;
419  RETURN
```

```
420  PRINT "A";
421  RETURN
422  PRINT "B";
423  RETURN
424  PRINT "C";
425  RETURN
426  PRINT "D";
427  RETURN
428  PRINT "E";
429  RETURN
430  PRINT "F";
431  RETURN
440  REM PRINT HEX ADDRESS
450  PRINT
455  REM CONVERT IT TO HEX
460  N = L/4096
470  IF N<0 N=-N
480  GOSUB 400+N+N
483  LET N= (L-N*4096)
486  GOSUB 400+N/256*2
490  GOSUB 400+(N-N/256*256)/16*2
495  GOTO 400+(N-N/16*16)*2
496  GOTO =GOSUB, RETURN
500  REM INPUT HEX NUMBER
501  REM FORMAT IS NNNNX
502  REM  WHERE "N" IS ANY HEX  DIGIT
505  N=0
509  REM INPUT LETTER OR STRING OF DIGITS
510  INPUT R
520  IF R=X RETURN
525  REM CHECK FOR ERROR
530  IF R>9999 THEN PRINT "BAD HEX ADDRESS
531  REM NOTE ERROR STOP ON LINE 530 (ON PURPOSE!)
535  REM CONVERT INPUT DECIMAL DIGITS TO HEX
```

```
540  IF R>999 THEN N=N*16
545  IF R>99 THEN N=N*16
550  IF R>9 THEN N=N*16
555  IF R>0 THEN R=R+R/1000*153+R/100*96+R/10*6
559  REM PICK UP NON-DECIMAL DIGIT LETTERS
560  IF R<0 THEN LET R=-R
565  REM ADD NEW DIGIT TO PREVIOUS NUMBER
570  LET N=N*16+R
580  GOTO 510
590  NOTE:  DON'T NEED END HERE


1000 TO RUN RANDOM NUMBER PROGRAM, TYPE "RUN"
1010 IT WILL TYPE 8 LINES THEN STOP
1020 TO RUN HEX DUMP PROGRAM, TYPE "GOTO 100"
1030 IT WILL ASK FOR INPUT.  TYPE 2 HEX ADDRESSES
1040 TERMINATED BY THE LETTER X,
1050 AND SEPARATED BY A COMMA
1060 THE PROGRAM WILL DUMP  MEMORY BETWEEN
1070 THOSE TWO ADDRESSES, INCLUSIVE.
1080 EXAMPLE:
1090 :GOTO 100
1100 DUMP: L.U/  A1BEX, A246X
1110 A1BE EE FF
1120 A140 00 11 22 33 44 55 66
1130 IF THE RANDOM NUMBER PROGRAM
1140 IS REMOVED, OR IF YOU TYPE IN
1150 :1 GOTO 100
1160 THEN YOU CAN GET THE SAME DUMP BY TYPING
1170 :RUN, A1BEX, A146X
1180 .
1190 NOTE THAT THIS PROGRAM DEMONSTRATES NEARLY
1200 EVERY FEATURE AVAILABLE IN  TINY BASIC 6800
```

```
10000  REMARK:  TO FIND OUT HOW MUCH PROGRAM SPACE
10001  REM... YOU HAVE LEFT, TYPE:
10002  LET P=USR (-8172, 38) - USR (-8172,36)
10003  PRINT P*256 + USR (-8172, 39) - USR (-8172,37)
10004  REM... OR COMBINE IT ALL INTO ONE PRINT.
```

```
:REM THIS IS A DATA LOGGING PROGRAM, USING THE 1 KHZ ON-BOARD
:REM REAL-TIME CLOCK TO MAINTAIN THE TIME OF DAY, AND
:REM MONITORING THE DATA ON THE PROM BURNER PIA (EASY ACCESS
:REM SOCKET); WHEN THE DATA CHANGES, THE NEW DATA IS PRINTED
:REM WITH THE TIME OF CHANGE.  AN INTERRUPT SERVICE ROUTINE
:REM IS STORED IN MEMORY LOCATIONS 0004-000F, USING THE USR
:REM FUNCTION (LINES 560-590).  LINE 550 STORES AN CLI INSTRUCTION
:REM WITH RTS IN LOCATIONS 0002-0003, AND LINE 600 SETS UP THE
:REM INTERRUPT VECTORE AND ENABLES THE INTERRUPTS BY JUMPING
:REM TO THE SUBROUTINE AT 0002.  LINE 620 SETS UP THE CONTROL
:REM AND THE DATA DIRECTION REGISTERS OF THE PIA.  THE DATA SAMPLED
:REM IS THE PROM DATA BUS, AND IS ACCESIBLE ON PINS 2-9 OF THE
:REM PROM BURNER SOCKET.  NOTE THAT THIS PROGRAM OCCUPIES NEARLY
:REM ALL OF THE RAM ON THE PROTO BOARD, BUT LINES 510-620 ARE
:REM EXECUTED ONLY TO SET UP MEMORY, AND MAY BE ENTERED WITHOUT
:REM LINE NUMBERS (DIRECT EXECUTION) TO SAVE SPACE.
:LIST

90 GOSUB 50
100 R=USR(I,X)
110 IF R<>D GOSUB300
120 IF U-T<1000 GOTOP
130 T=T+1000
140 S=S+1
150 IF S<60 GOTOP
160 S=0
170 M=M+1
180 IF M<60 GOTOP
190 M=0
200 H=H+1
210 GOTOP
300 PRINT
310 D=R
320 N=0
330 R=R*256
```

```
340 B=0
350 IF R<0 B=1
360 PRB;
380 N=N+1
390 R=R+R
400 IF N<8 GOTO 340
410 PR" ";H;":";M;":";S;
420 RETURN
500 P=100
510 S=-8168
520 I=S-4
550 A=USR(S,2,14)+USR(S,3,57)
560 A=USR(S,4,183)+USR(S,6,196)
570 A=USR(S,7,124)+USR(5,8,0)+USR(S,9,171)
580 A=USR(S,10,38)+USR(S,11,3)
590 A=USR(S,12,124)+USR(S,13,0)+USR(S,14,170)
600 A=USR(S,15,59)+USR(S,-8,0)+USR(S,-7,4)+USR(2)
610 X=-1082
620 A-USR(S,X-1,7)+USR(S,X+1,0)+USR(S,X,0)+USR(S,X+1,4)
630 D-USR(I,X)
640 PR "TIME (H,M,S)";
650 INPUT H,M,S
660 LET T=U
680 RETURN

:RUN
TIME (H,M,S)? 3,17,50

00000000 3:17:59
00100000 3:18:10
00000000 3:18:15
00000001 3:18:16
00000000 3:18:30
10000000 3:18:31
```